IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of
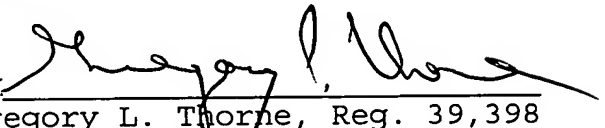
KARL J. WOOD ET AL

Serial No.: 10/077,062

Filed: FEBRUARY 15, 2002

Title: APPARATUS

Commissioner for Patents
Washington, D.C.   20231
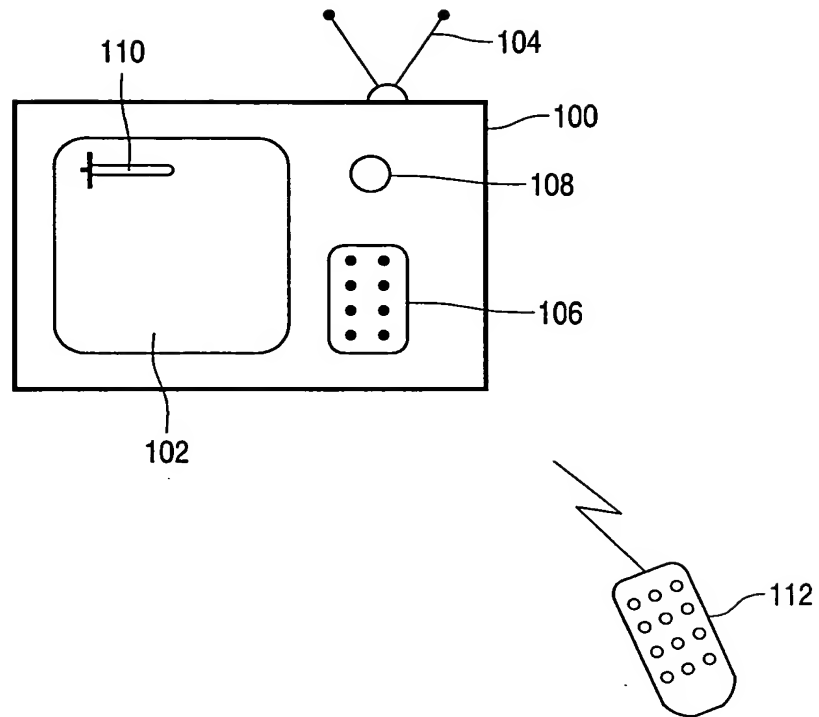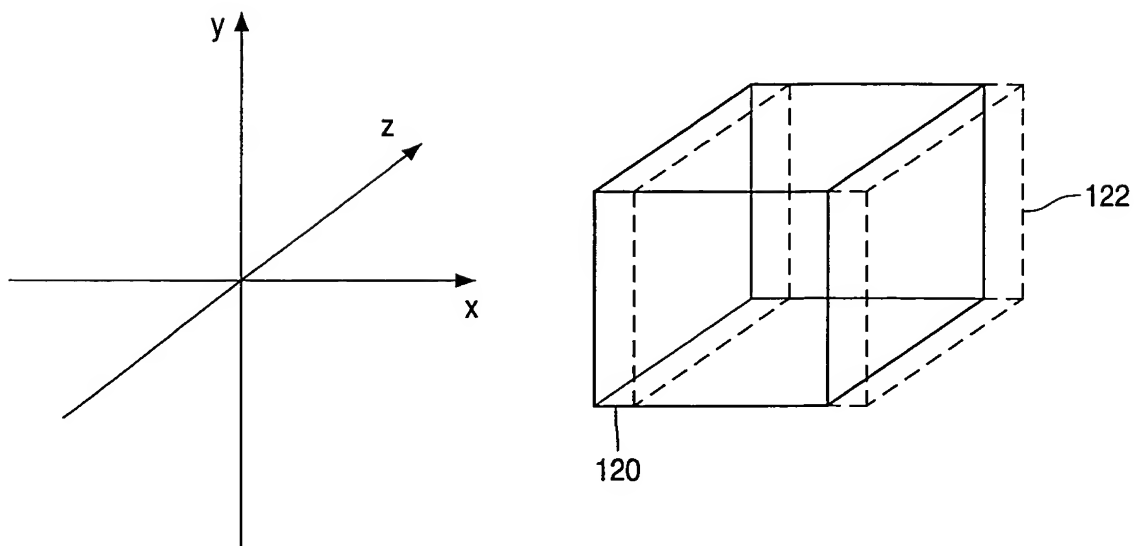
Atty. Docket

PHGB 010035

Group Art Unit: 2613

LETTER TO OFFICIAL DRAFTSMAN

Sir:

    Enclosed are (2) TWO sheets of formal drawings
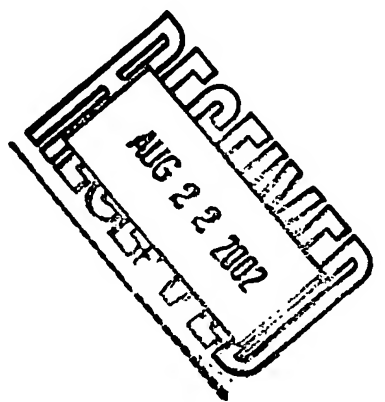
for filing in the above-identified application.

Respectfully submitted,

By_____,
Gregory L. Thorne, Reg. 39,398
Senior Patent Counsel
(914) 333-9665

RECEIVED
AUG 2 2 2002

S:\TH\mv01tha0.ec0.doc

FIG. 1

FIG. 2

FIG. 3

FIG. 4

2123
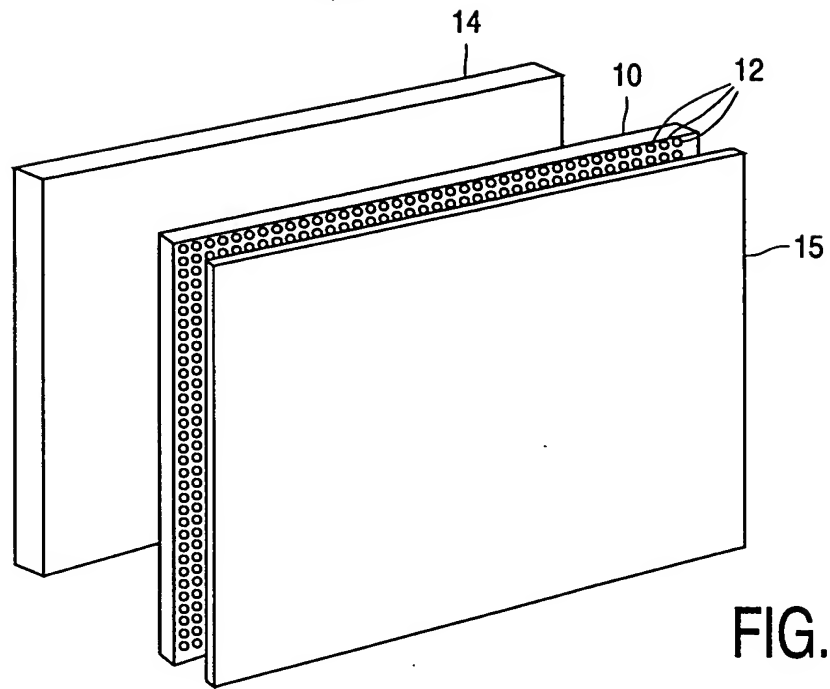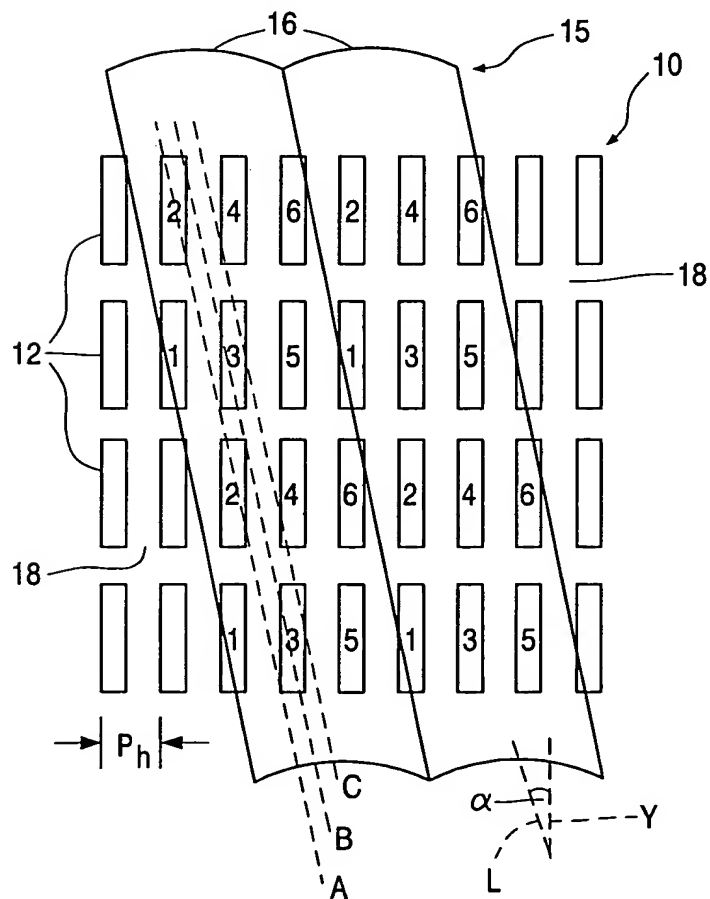
Communication Link Requirements

Link
1 — Design

Bit Error Rate
Performance Estimation

subsystem model descriptions

Algorithm
2 — Design

Aqcuisition Performance
Tracking Stability
Spectral Properties
Multiaccess

Tx → Chan → Rx

Architecture
3 — Design

Active Area Throughput
Estimated Power
Estimated Timing
Bittrue behavior

Map
DF ⊗
DF ⊗
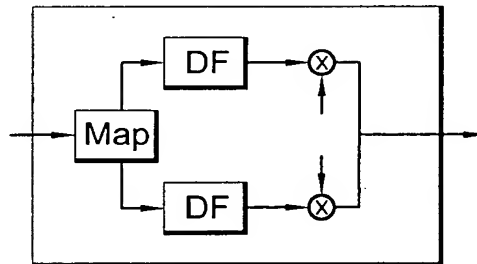
Circuit
4 — Design

Power
Routing Capacitance
Netlist Errors
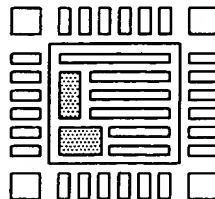
FIG. 1A
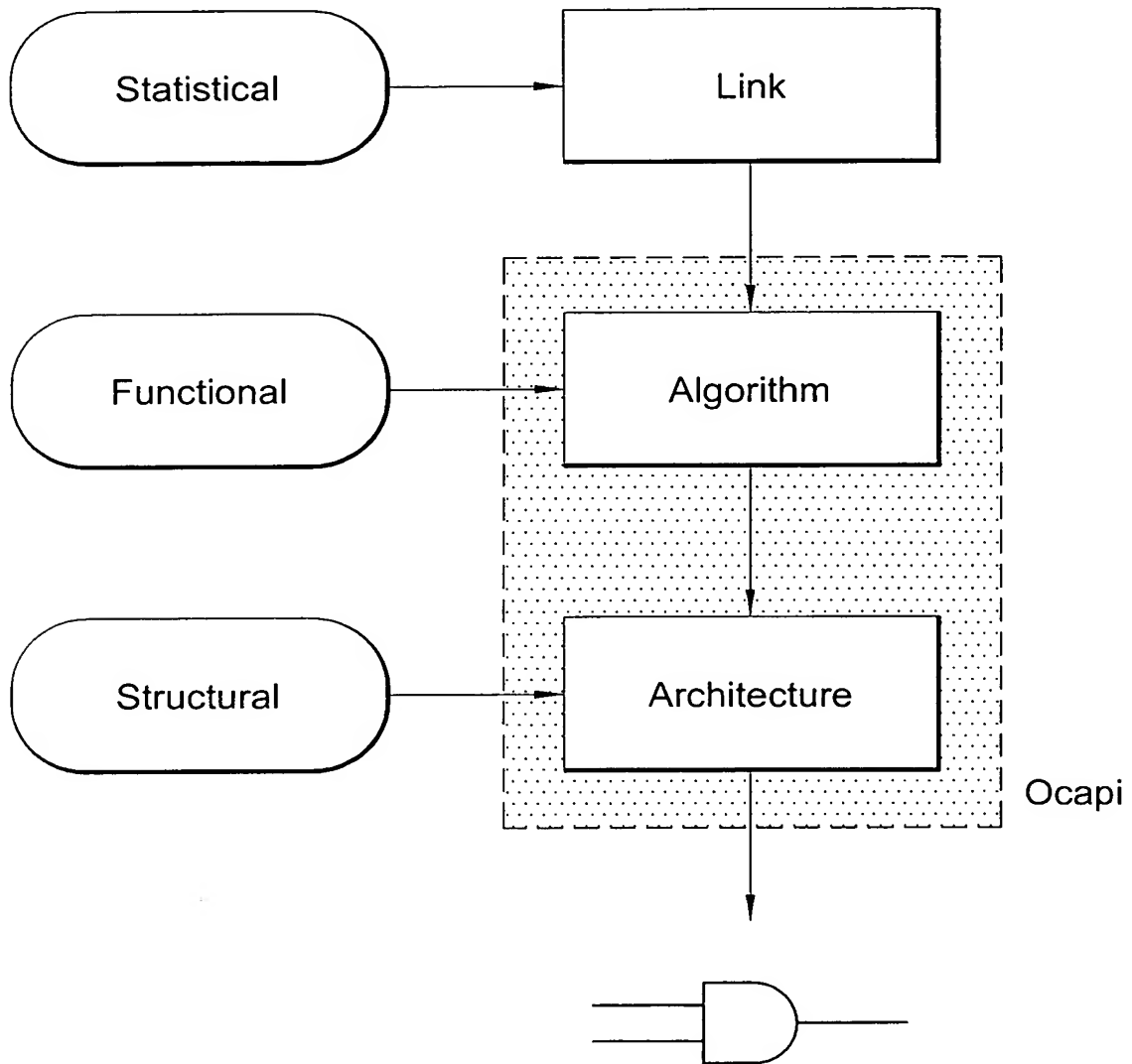
FIG. 1B

FIG. 1C

FIG. 1D

RECEIVED

AUG 1 5 2002

Technology Center 2100

FIG. 2

A DESIGN APPARATUS AND A METHOD FOR GENERATING AN IMPLEMENTABLE DESCRIPTION OF...

Schaumont, et al.

Appl. No.: 09/237,549   Atty Docket: IMCE65.001CP1

RECEIVED

AUG 1 5 2002

Technology Center 2100

FIG. 3

FIG. 4

FIG. 5

FIG. 6

707

queue

token

708

queue

706

queue

Simulation & Modeling

701

sfg1

registered
signal

702

sfg2

704

s0

s1

705

clk

703

sfg3

registered
signal

fsm

actor

709 queue

datapath sfg

710

fsm-to-sfg

711

sfg1   sfg2   sfg3

sfg
fsm

Synthesis

712

bus   strobe

conditions

data
path

clk

mode

fsm

bus   strobe
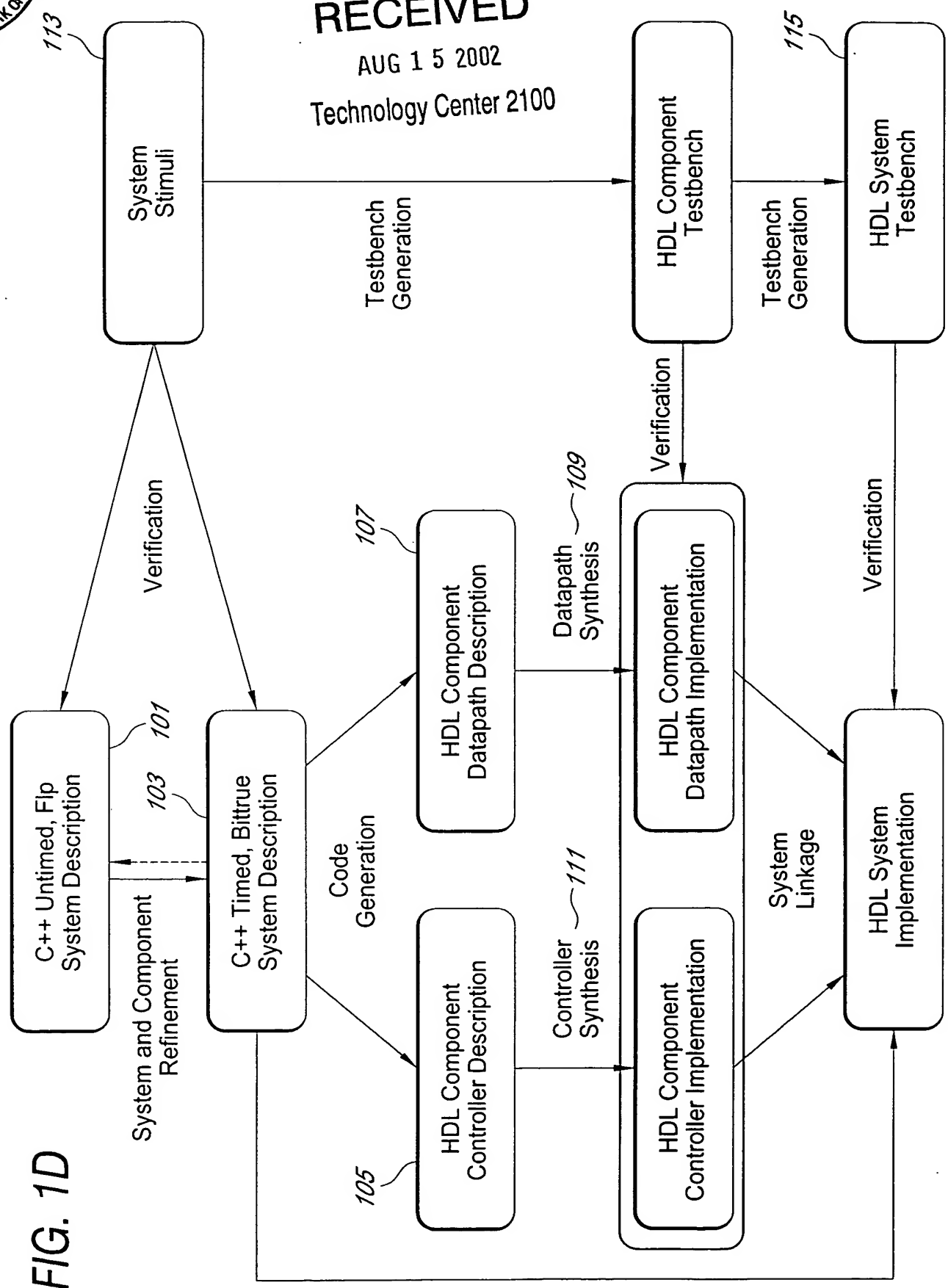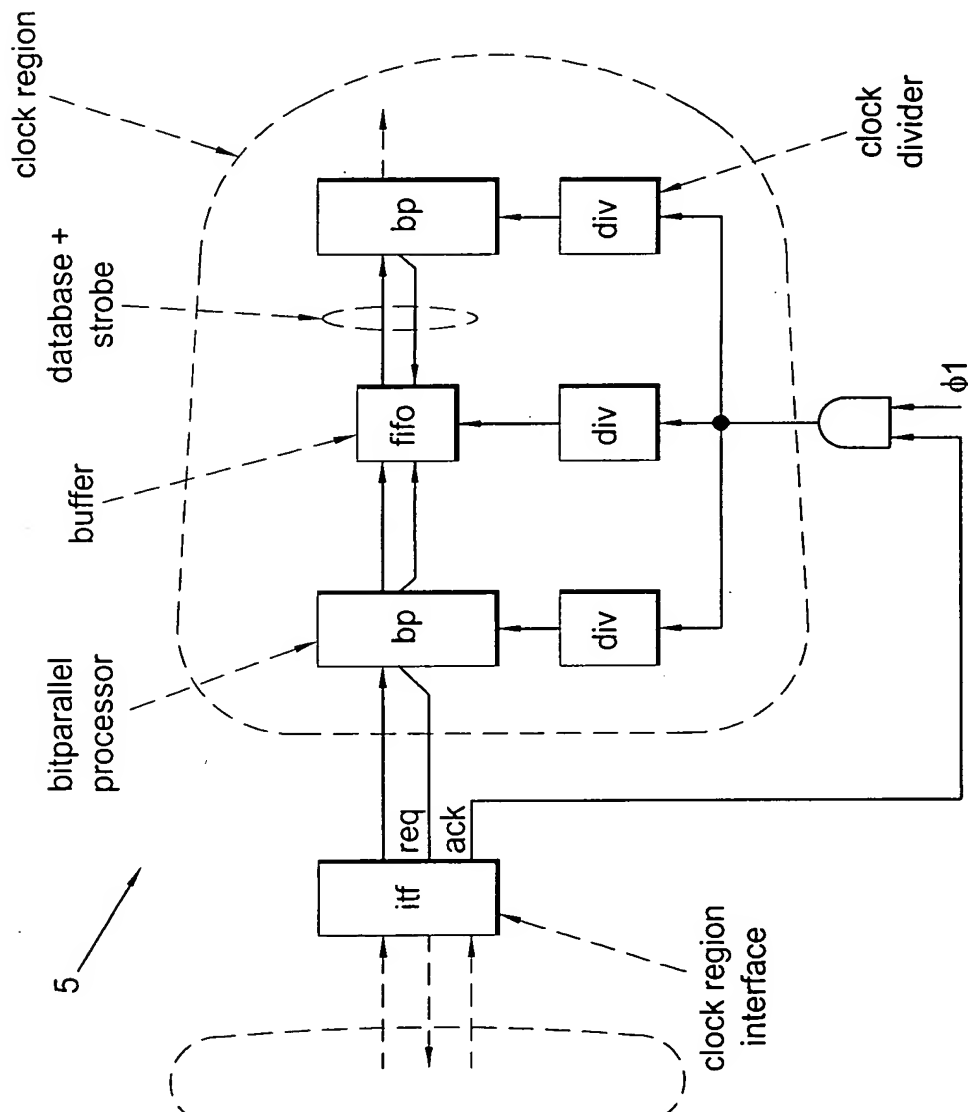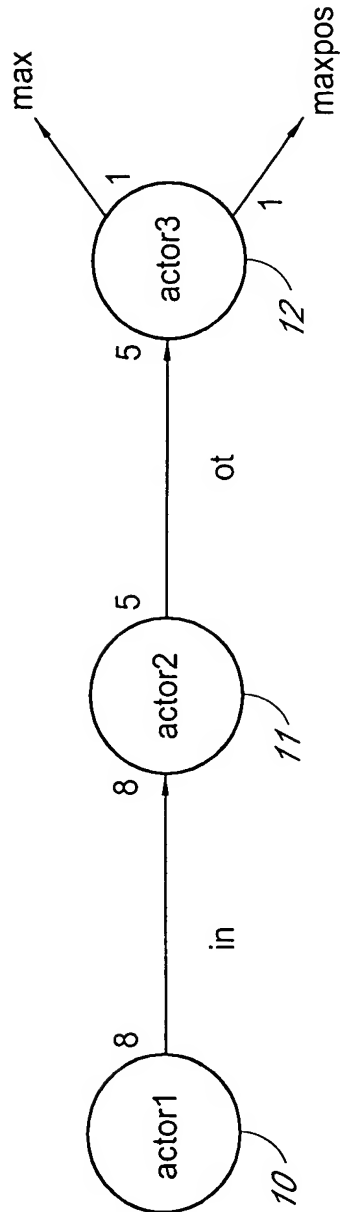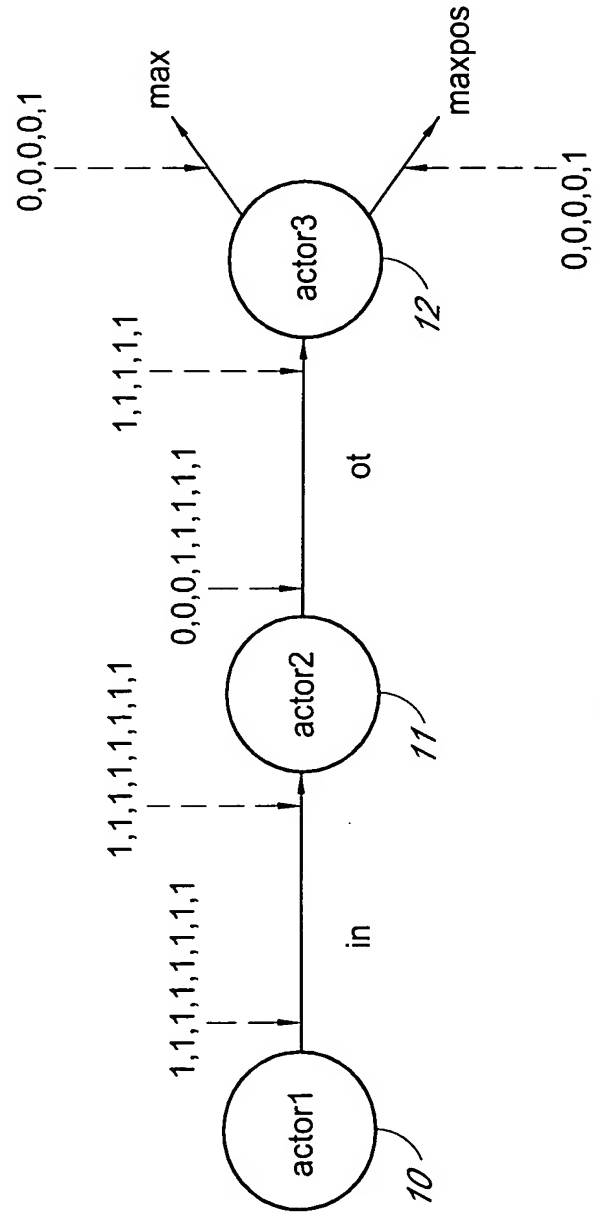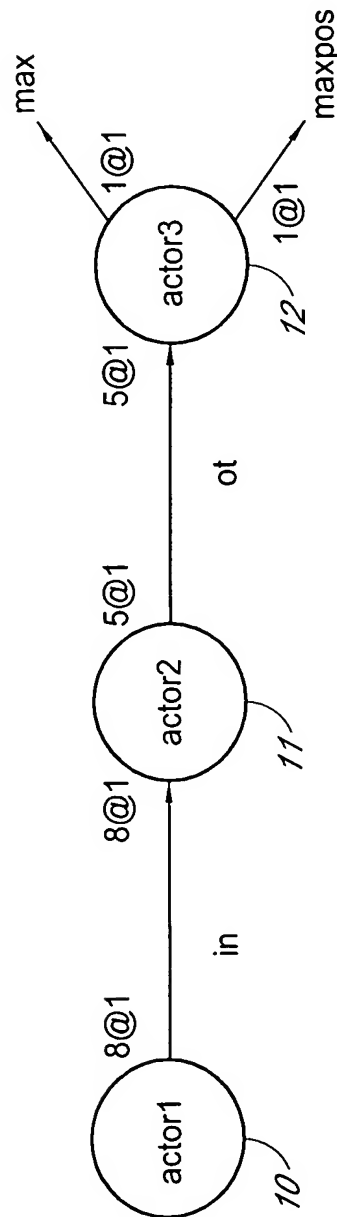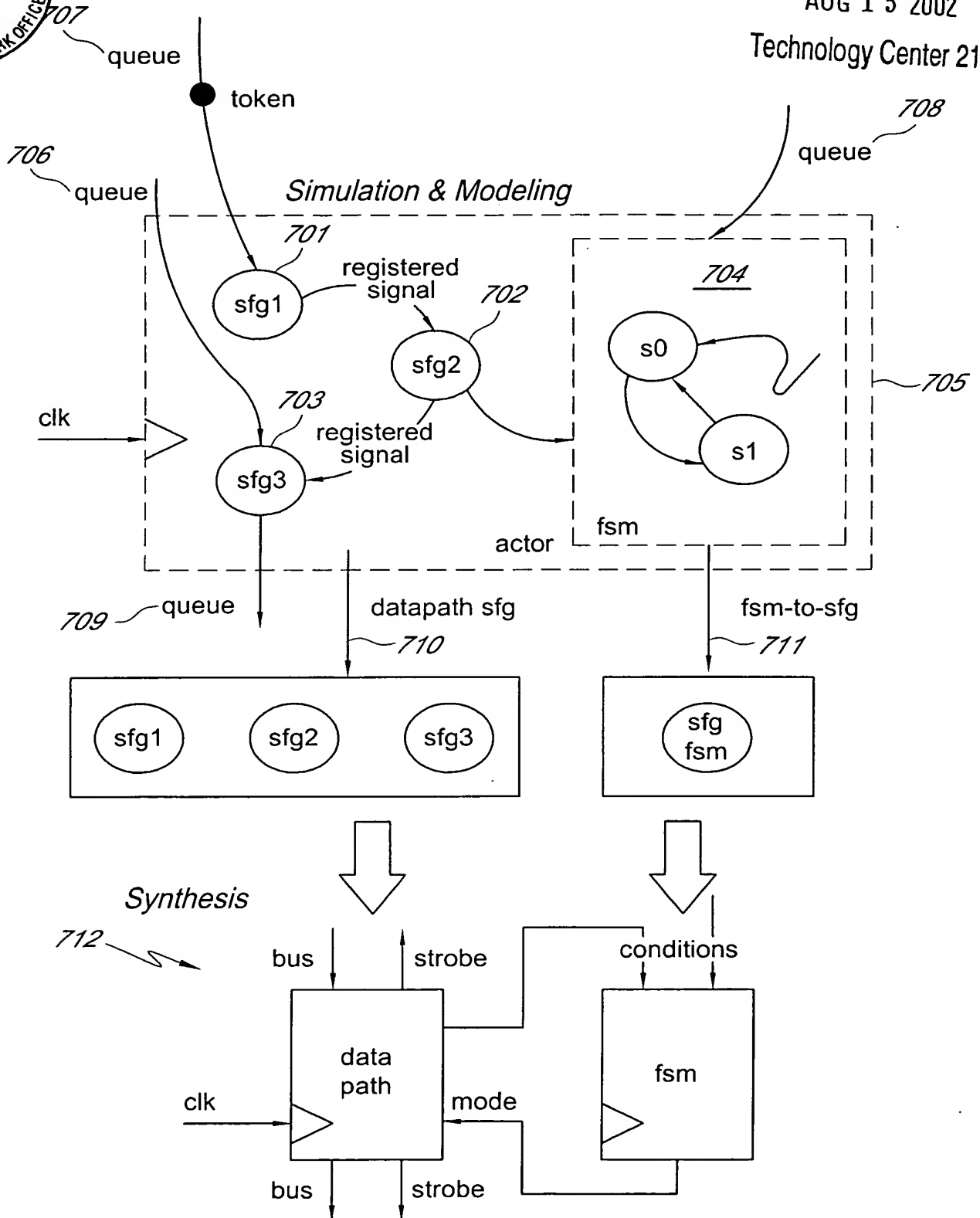
FIG. 7

FIG. 8

TRAVEL DELAY=5
TOKEN CONCURRENCY=1
TOKEN LATENCY=5

FIG. 9A

902
901

EACH 5 CYCLES

TRAVEL DELAY=5
TOKEN CONCURRENCY=1
TOKEN LATENCY=1

FIG. 9B

903
904   905   906   907   908

EACH CYCLE
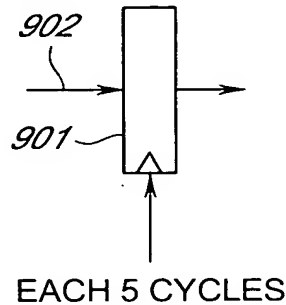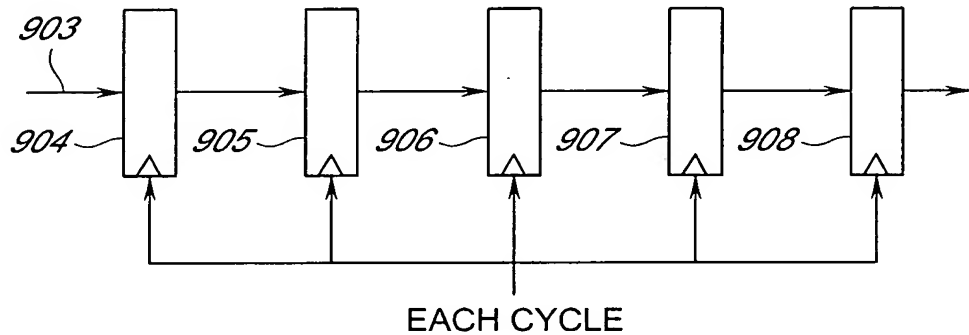
TRAVEL DELAY=5
TOKEN CONCURRENCY=2
TOKEN LATENCY=1

FIG. 9C
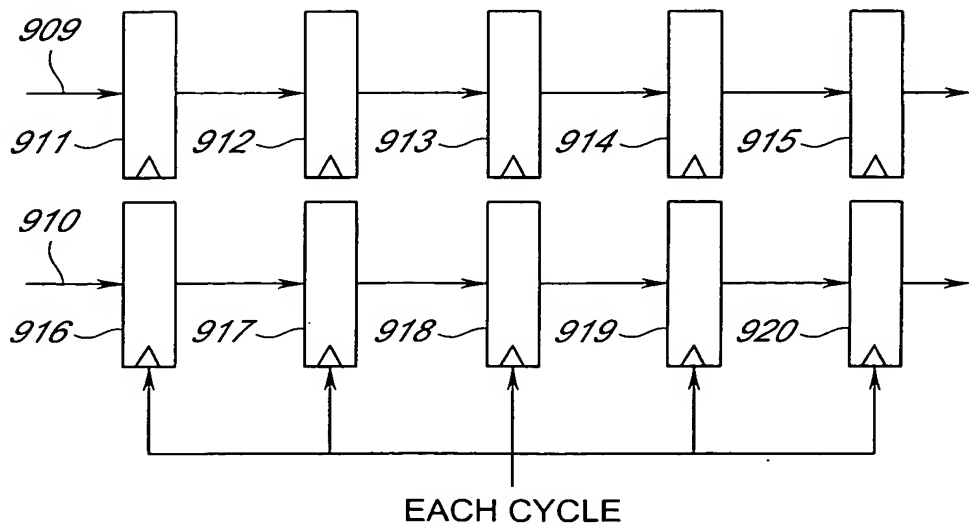
909
911   912   913   914   915

910
916   917   918   919   920

EACH CYCLE

token age   queue age

actor   ● token   queue   →

iteration delay   travel delay

b
  ↘
    (+) → c
  ↗
a

operation

Stage 1
if actor fires:
    for all actor output queues:
        queue age = queue age + iteration delay

Stage 2
if operator is executed:
    output token age = max (input token ages)

if token enters queue:
    token age = token age + travel delay

Stage 3
CHECK EXPECTATION:
    token age at output of queue
            > queue age
IF SUCESS:
    queue age = token age
IF FAIL:
    warn user
    token age = queue age
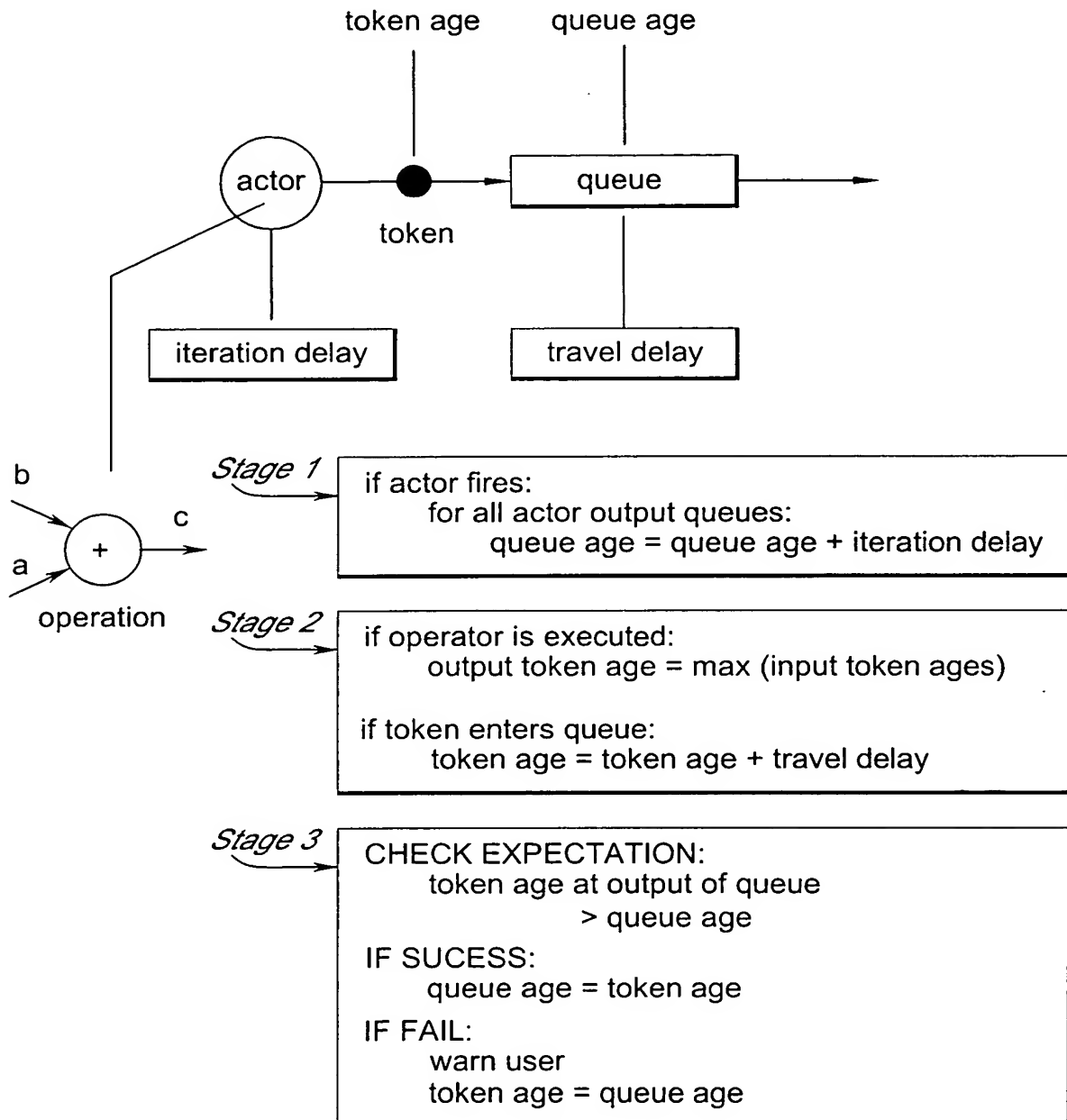
FIG. 10

```
dfix  T_sample(8, 6);
dfix  T_acc   (8, 6);
dfix  T-bit   (1, 0, ns);
double hardwired_coef = { 0.5, 0.2, -0.3, 0.15 };

fsm correlator: :define(clk & _ck)
{

    sig_array   coef         (4,  ck,  T_sample);
    sig_array   sample       (4,  ck,  T_sample);
    sig         accu         (ck,  T_accu        );
    sig         sample_in    (T_sample           );
    sig         coef_in      (T_sample           );
    sig         corr_out     (T_sample           );
    sig         load         (ck, T_bit          );
    sig         load_ctr     (T_bit              );


    sfg initialize_coefs;
    for (i = 0;  i < 4; i++)
        coef[i] = W(T_sample, hardwired_coef[i] );

    sfg load_coef_0;
    input(coef_in);
    coef[0] = in_coef_in;

    sfg correl_1;
    accu  = cast(T_acc, coef[0] * sample[0] + coef[1] * sample[1]);

    sfg correl_2;
    corr  = accu + cast(T_acc,  coef[2] * sample[2] + coef[3] * sample[3] );
    output(corr);

    sfg read_sample;
    input(sample_in);
    for (i = 3; i >=0; i--)
        if (i)
          sample[i] = sample[i-1];
        else
          sample[i] = sample_in;

    sfg read_control;
    input(load_ctr);
    load = load_ctr;

    fsm myfsm;
    initial rst;
    state phase_1
    state phase_2
    rst        <<  always     <<  initialize_coefs    <<  phase1;
    phase1 <<   always     <<  read_control
                           <<  correl_1            <<  phase2;
    phase2 <<   !cnd(load)  <<  correl_2
                           <<  read_sample         <<  phase1;
    phase2 <<    cnd(load)  <<  correl_2
                           <<  read_sample
                           <<  load_coef_0         <<  phase1;
    return mysfm;
}
```
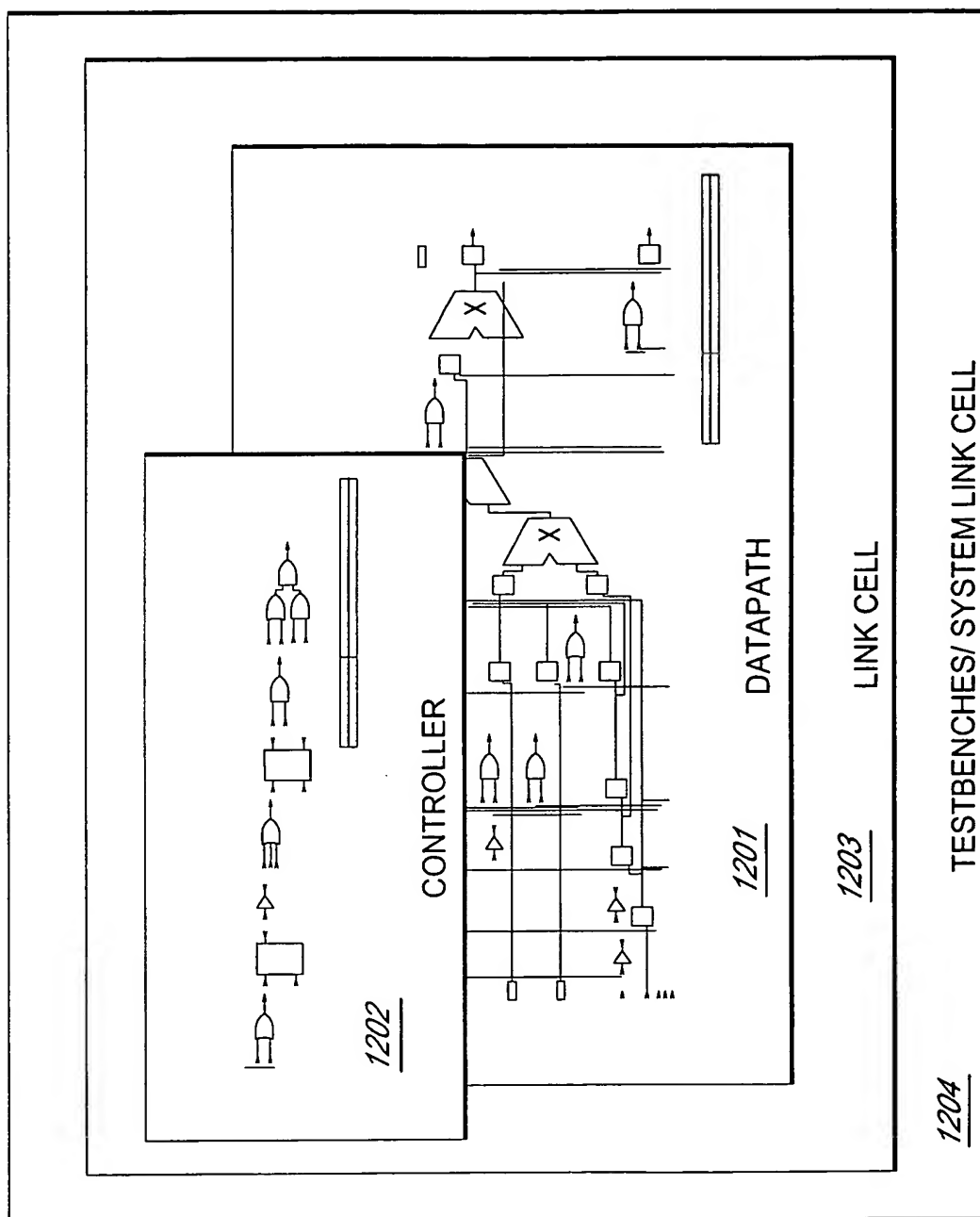
FIG. 11

CONTROLLER

1202

DATAPATH

1201

LINK CELL

1203

TESTBENCHES/ SYSTEM LINK CELL

1204

FIG. 12

FIG. 13

FIG. 14

~30

reset processing

hold_request /
hold_on

!hold_request /
lookup

hold_request /
wait_lookup

execute   hold

!hold_request /
hold_lookup

~31

SFG lookup   datapath instruction busses

1503

1502   lookup Table

PC

hold_request

1501

SFG hold_on

1507

PC   hold_pc   1508

hold_request

SFG wait_lookup   datapath instruction busses

1509   "nop"
        "nop"
        "nop"

hold_request

1504

SFG hold_lookup   datapath instruction busses

1510   hold_pc   lookup Table

hold_request

1505

1511

FIG. 15

A DESIGN APPARATUS AND A METHOD FOR GENERATING AN IMPLEMENTABLE DESCRIPTION OF ...

Schaumont, et al.

Appl. No.: 09/237,549 Atty Docket: IMEC65.001CP1
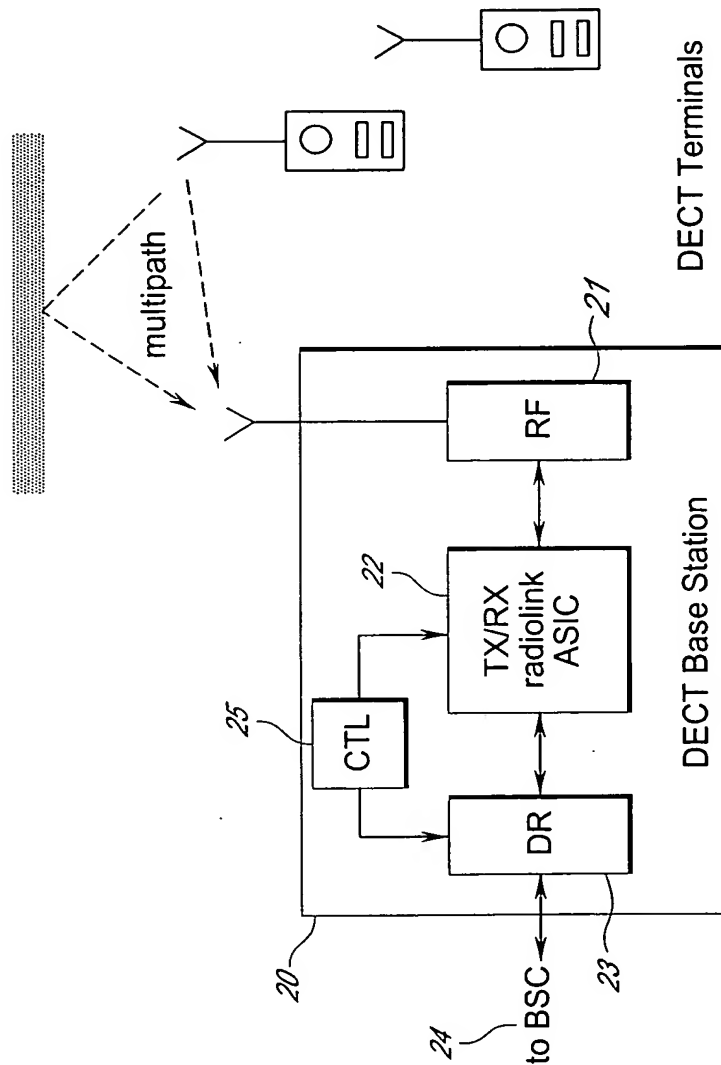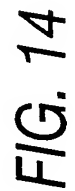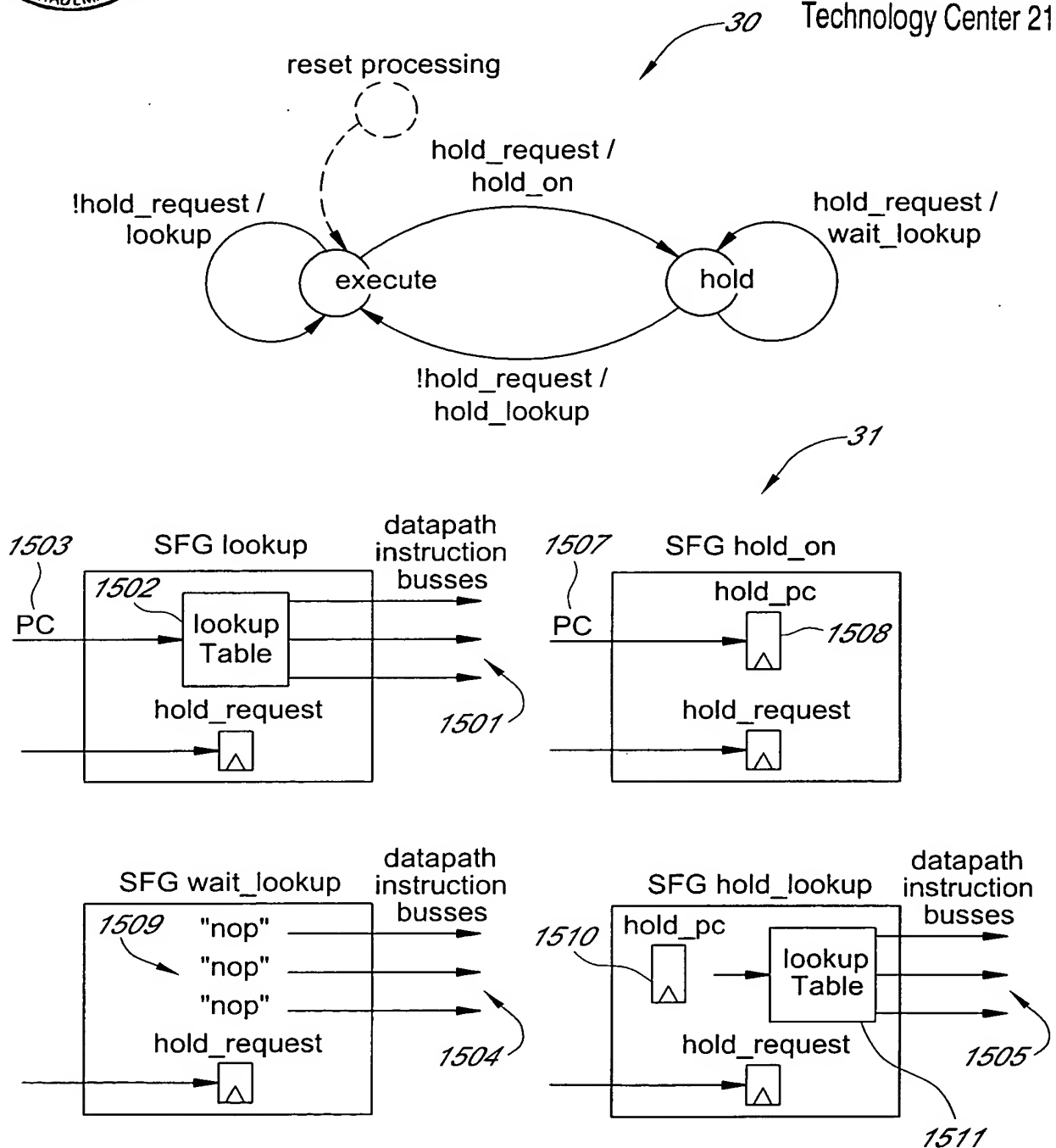
Sig Class

```
class sig {
   Value value;
   char  *name;
public:
   sig(value v);
   sig operator + (sig v);
   virtual Value simulate ();
   virtual void gen_code (ostream &os);
};

sig sig: :operator + (sig v) {
   sigadd s;
   add.left  = &v;
   add.right  = this;
   return add;
}

Value sig: :simulate() {
   return value;
}

sig: :gen_code (ostream &os) {
   os << name;
}
```

Derived Operator Class

```
class sigadd  :  public  sig {
   sig  *left;
   sig    *right;
public:
   Value simulate();
    void gen_code (ostream &os);
};

Value sigadd: :simulate() {
   return  left->eval()  +
            right->eval();
}

sigadd: :gen_code (ostream & os) {
   os  <<  left->cg()
   <<  " + "
   <<  right->cg();
}
```

FIG. 16

```
sig a, b, c, d;
b = a + 3;
d = (b + c) << 3;
```
40

sig

sigadd

FIG. 17

41

1801

!eof/sfg3

_/sfg1

S0

S1

1802
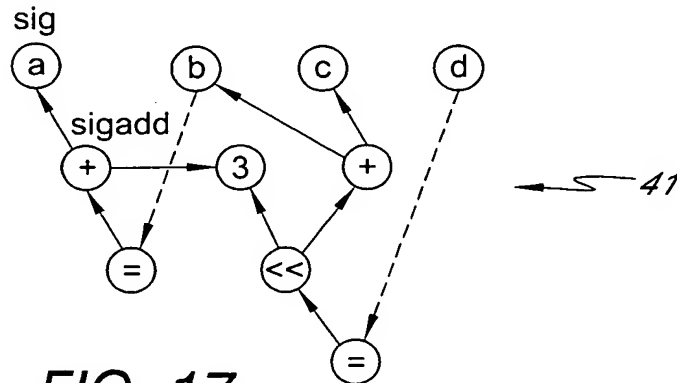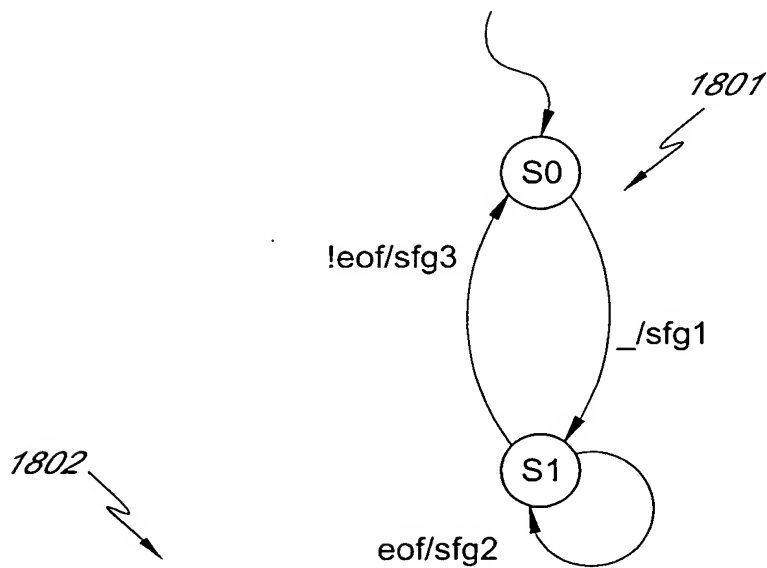
eof/sfg2

```
fsm f;
initial s0;
state s1;

s0 << allways        << sfg1 << s1;
s1 << cnd(eof)       << sfg2 << s1;
s1 << !cnd(eof)      << sfg3 << s0;
```

FIG. 18

RECEIVED

AUG 1 5 2002

Technology Center **2100**

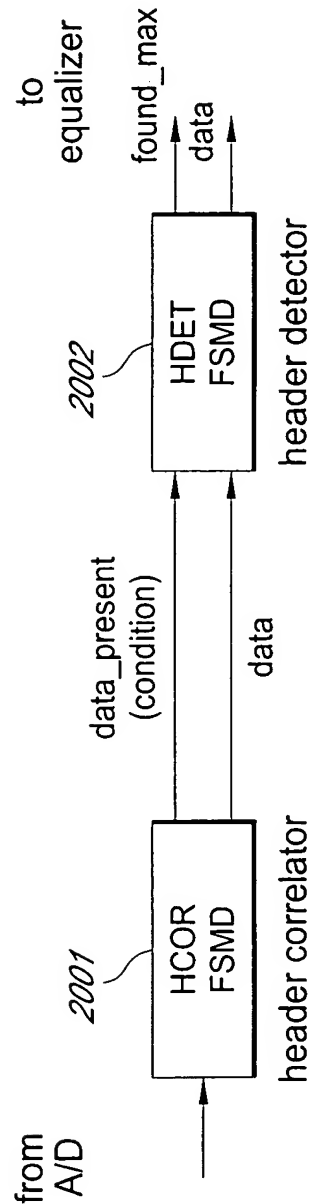FIG. 19

RECEIVED

AUG 1 5 2002

Technology Center 2100

FIG. 20

ONE CLOCK CYCLE

| token production | evaluation | register update |

clk

Component 1 ~60

FSM

SFG

①

System Interconnect

②b

Component 2 ~61

FSM

+ 1 SFG

②a

Component 3 ~62

```
int c::run() {
    // firing rule
    ...
    // behavior
    ...
}
```

FIG. 21

2201 — **C++ High Level System Description**

↓ Refinement

2202 — **C++ Timed, Bittrue System Description**

2203 — Compile Execute ↓

2204 — **FSMD Data Structure**

Interpreted Simulation — 2205

C++ Code Generation ↙   **HDL Code Generation** ↘

2206 — **C++ RT Description**   2208 — **HDL Components Descriptions**

Compile 2207 ↓   2209 / HDL Simulation & Synthesis ↓

Compiled Code Simulation   **Hardware Implementation**

FIG. 22

FIG. 23

GEN

gen
216

TX

rnd
209

x

tuple
210

diffenc
211

map
212

shape
213

da
214

215

CX

fir
217

noise
218

RX

lmsff
219

constel

demap
220

diffenco

detuple
221

x

derand
222

FIG. 24